**MARC FINDELI** : marc@findeli.com

July 2001



# Peer-to-Peer (P2P) Networking

Lehrstuhl für Kommunikationsnetze an der Technischen Universität München

# Peer-to-Peer (P2P)

MARC FINDELI

## 1 Introduction

The success and notoriety of the Napster online music-file-sharing program has raised the awareness of a "new" technology that has actually been around for a while: peer-to-peer (P2P) computing. P2P is by no means a new idea. It has its roots in LAN file sharing, which became popular during the mid-1970th. The difference in the recent focus on P2P seems to be that it has finally caught the imagination of people building practical systems.

With P2P, computers share data and resources, such as spare computing cycles and storage capacity, via the Internet or private Networks, irrespective of their location on the network. P2P encourages users to form networks devoted to an interest or task.

P2P provides the user kinds of enhanced newsgroups. P2P gives access to a lot of interesting services such as:

- *instant messaging*
- *file sharing (Gnutella, Napster, ...)*
- *tele-conferences*
- *collaborative environment (seti@home, ...)*
- *voice communication (IP-telephony, ...)*

Sharing music files is one of the most controversial application of P2P technologies. Napster and Gnutella have establish two different P2P protocols. These two successful and mostly different technologies can be extended to a lot of other fields.

Taking for example the field of collaborative environment, an office worker telecommuting from home can, with the appropriate P2P infrastructure, transparently access files and information that reside within the office network. In fact, P2P computing erases the distinction between the "home" network and the "office" network; it forms a virtual network which could integrate all of your disparate devices into a whole cohesive whole.

In another circumstance, this office worker ought to locate a college, irrespective of his location in the world, to start a voice- or video-communication over internet. In this situation he should use a P2P protocol such as *SIP* to locate the college.

## 2 Peer-to-Peer Networking

Most Internet services such as the world-wide-web are implemented in *client/server* processing model. The client/server model refers to a two-node relationship characterised by fixed capabilities, where client and server are confined to their roles. Clients, nodes in relation to edges, initiate communications but do not accept communications initiated elsewhere. Clients don't process requests of other clients. This is the function of servers. On the other hand servers can't initiate communications with clients.

Unlike this asymmetric scheme, the peer-to-peer network concept defines a network at the *application layer* where a node works as client and server at the same time. Hosts can communicate directly and manage computing tasks without using central servers whereas they are performing as a fulfil node of the P2P network. In a world where network access is universal, bandwidth is plentiful, and processor cycles are inexpensive, peering, also among physical unequals, is getting more and more desirable.

Thus Peer-to-Peer refers to any relationship in which, autonomous hosts, the peers, interact as equals. The communications between these corresponding systems specify the P2P communication. The peers provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other peers, check for matches against their local data set, and respond with applicable results. In the client/server architecture these tasks are strictly separated between servers and clients.

To be accurate, it is important to remind that the P2P-data-packets are transmitted with the TCP/IP protocols which operate in lower OSI layers and still implement the client/server processing model.

# 3 File Sharing

A few Internet-based applications have hinted at P2P's potential; for example the *file-sharing* applications such as *Napster* and *Gnutella*. P2P permits direct sharing of documents, multimedia, and other files between computers. Instead of storing the files on central servers, the files live on users' machines. In file-sharing models each machine, a peer, offers a portion of its own file system or asks for a remote file.

To manage this, the user could choose one of the two most popular P2P-applications that are Napster and Gnutella. In the next chapters the two protocols are going to be discussed, with all their similarities and differences.

As a consequence of the popularity and the open-source aspect of both protocols, a lot of clones are circulating in the internet (BearShear, OpenNap, ...). However the fundaments are identical.

## 3.1 Napster Protocol

Due to its ease of use Napster has become very popular. Though in the next months sharing mp3 via Napster won't be anymore possible in its current form. Because Napster is not an open source application, it was only possible to build up a similar application in revealing the Napster protocol by reverse-engineering. The project OpenNap has made it possible to run a Napster server on many platforms without using the original Napster application and the index server. In this context it is also not a problem to build up diverse applications which use a Napster similar protocol to share any type of files with the habitual ease of use.

Napster works with a central server which maintains an index of all the mp3 files of the peers. To get a file you have to send a query to this server which sends you the port and IP address of an client sharing the requested file. With the Napster application it is now possible to establish a direct connection with the host and to download a file.

Napster is not a pure P2P application. It has a hybrid characteristic. In fact, the Napster network is characterised by highly centralised topology [Figure1]. The file searching protocol uses a client/server model using a central index server; on the other hand the file transfer and the direct browsing between users run on a state-of-the-art P2P architecture. These two last applications occur directly between the Napster hosts without passing the server; but, to be accurate, it is important to know that Napster hosts are relating every file transfer or browsing even if the host is acting like a server.

Contrarily to Gnutella, the Napster protocol uses a lot of different types of messages. Every state of the hosts, acting like clients towards the server, is related to the central Napster server. Thus the Napster protocol makes anonymity impossible. On the other hand, this complex protocol makes a lot of services possible such as:

- *Instant Messaging:* sending public or private messages to other users; creating and joining channels of shared interests
- *Creating Hotlists:* notifying when users of your own hotlist sign on or off the server
- *List of ignored User*

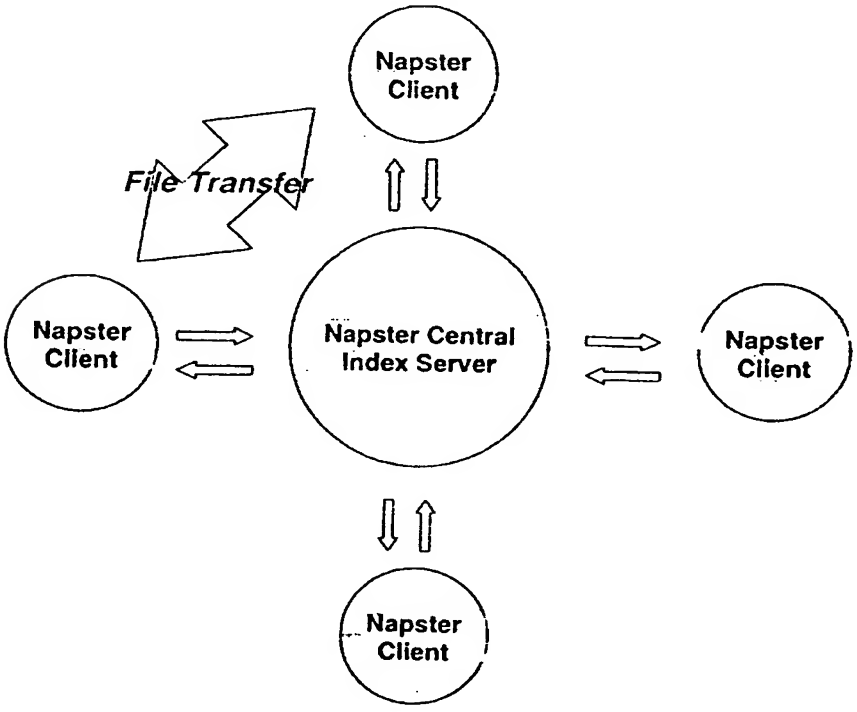


**FIGURE 1: Napster Network topology**

## Napster Messages Data Structures

Each message to/from the Napster central server is of the following form:

| <Length> 2 Bytes | <Function> 2 Bytes | Payload n Bytes |
|---|---|---|

- ***Length:*** specifies the length of the payload.
- ***Function:*** defines the message type of the packet (see next paragraph)
- ***Payload:*** this portion of the message is a plain ASCII string

Every block of header and payload is separated by "blanks" which make the synchronisation of the incoming messages possible. Most of blocks have no fixed length. The blanks make separation of data blocks in incoming bit streams possible.

## Function IDs

The Napster protocol uses nearly up to 1000 different type of messages. In the following descriptions the IDs will be written in brackets: e.g. SEARCH(200).

## Initialisation

A registered Napster host, acting like a client, sends to the server a **LOGIN(0x02)** message with the following format:

| <Nick> | <Password> | <Port> | "<Client-Info>" | <Link-Type> |
|---|---|---|---|---|

- ***Nick & Password:*** identify the user
- ***Port:*** the port the client is listening on for data transfer.
- ***Client-Info:*** is a string containing the client version info.
- ***Link-Type:*** is a integer indicating the client's bandwidth (0: unknown, 1: 14.4 kbps, 2: 28.8 kbps, 3: 33.6 kbps, 4: 56.7 kbps, 5: 64k ISDN, 6: 128k ISDN, 7: Cable, 8: DSL, 9: T1, 10: T3 or greater)
- ***Example:*** lkn retspan 6699 'hap v0.8"9

The host's IP address hasn't to be added to the message. Indeed the server can extract it automatically from the TCP packet in which the message is packed for the transmission.

An unregistered host, sends a **NEW USER LOGIN(0x06)** which is similar to the format of LOGIN(2), with the addition of the email address on the end.

The server sends a **LOGIN ACK(0x03)** to the client after a successful login. If the nick is registered, the email address given at registration time is returned. If the nick is not registered, a dummy value is returned.

With the **CLIENT NOTIFICATION OF SHARED FILE(0x64)** message the client sends successively all the files it want to share.

| "<Filename>" | <MD5> | <Size> | <Bitrate> | <Frequency> | <time> |
|---|---|---|---|---|---|

- ***MD5:*** hash value of the shared file. The MD5 (Message Digest 5) algorithm produce a 128-bit 'fingerprint" of any file. It is nearly computationally infeasible to produce two messages having the same hash value. The MD5 algorithm is intended to provide any user the possibility to secure the origin of his shared file, even if the file is laying on drives of other Napster users. (For more information about the MD5 algorithm see: http://www.faqs.org/rfcs/rfc1321.html .)
- ***Size:*** file size in bytes
- ***Bitrate:*** bit rate of the mp3 in kbps
- ***Frequency:*** sample rate of the mp3 in Hz
- ***Time:*** in seconds

## *File Request*

The downloading client will first issue either a **SEARCH(0xC8)** or **BROWSE(0xD3).** The first message has the following format:

| | |
|---|---|
| [FILENAME CONTAINS "Artist Name"] | [MAX_RESULTS <Max>] |
| [FILENAME CONTAINS "Song Name"] | [LINESPEED <Compare> <Link-Type>] |
| [BITRATE <Compare> "<BR>"] | [FREQ <Compare> "<Freq>"] |

- ***Max:*** maximal number of results
- ***Compare:*** is one of the following, "AT LEAST" "AT BEST" "EQUAL TO"
- ***Link-Type:*** see 0x02 for description
- ***BR:*** bitrate in kbps
- ***Freq:*** sample frequency in Hz
- ***Example:*** FILENAME CONTAINS "superband" MAX_RESULTS 128 FILENAME CONTAINS "supersong" LINESPEED "EQUAL TO" 10 BITRATE "AT LEAST" "128" FREQ "44100"

The artist name and the song name are treated the same by server. Thus it is possible to add some more search criterias with adding to the message some FILENAME CONTAINS blocks. Unfortunately the Napster server does not uses the ID3 of mp3 files in its search criteria.

The payload of the **BROWSE(0xD3)** message does only contains the <nick> of the host. It requests a list of the host's shared files.

The server answers respectively with a **SEARCH RESPONSE(0xC9)** or a **BROWSE RESPONSE(0xd4)** with the formats:

| | | | | | |
|---|---|---|---|---|---|
| "<Filename>" | <MD5> | <Size> | <Bitrate> | <Frequency> | <Time> |
| <Nick> | <IP> | <Link-Type> | | | |

- ***MD5:*** hash value of the requested file
- ***Size:*** file size in bytes
- ***Bitrate:*** bit rate of the mp3 in kbps
- ***Frequency:*** sample rate of the mp3 in Hz
- ***Time:*** specify the length of the file
- ***Nick:*** identify the user who shares the file
- ***IP:*** 4 Bytes integer representing the IP address of the user with the file.
- ***Link-Type:*** see LOGIN message

### *Download Request*

To request a download, a **DOWNLOAD REQUEST(0xCB)** message is sent to the server. The client requests to download <filename> from <nick>. This message has the following payload format:

| <Nick> | "<Filename>" |
|---|---|

- ***Example:*** supernap "supersong.mp3"

The server will answer with a **DOWNLOAD ACK(0xCC)** contaning more information about the file (Linespeed, Port Number, ...):

| <Nick> | <IP> | <Port> | "<Filename>" | <MD5> | <Link Type> |
|---|---|---|---|---|---|

### *File Transfer*

At this point the hosts doesn't send Napster messages to the central server. The host requesting the file makes a TCP connection to the data port specified in the 0xCC message from the server. To request the file the client wish to download, the client sends the following HTTP – messages: a string 'GET" in a single packet and a message with the format:

| <MyNick> | "<Filename>" | Offset |
|---|---|---|

- ***MyNick:*** the client's nick
- ***Offset:*** the byte offset in the file to begin the transfer at. It is needed to resume prior transfer.

The remote host will then return the file size and, immediately following, the data stream. The direct file transfer between Napster hosts uses a P2P architecture.

Once the data Transfer is initiated, the downloader should notify the server that they are downloading a file by sending the **DOWNLOADING FILE(0xDA)** message. Once the transfer is complete, the client send a **DOWNLOAD COMPLETE(0xDB)** message.

## 3.2 Gnutella Protocol

Gnutella arose like an independent project from the programmers Justin Frankel and Tom Pepper, founders of the Nullsoft company, outstanding to be the owner of the Winamp product. On the 14th march 2000, in the Slashdot web site (http://www.slashdot.com) was published, in beta phase, a version of the Gnutella program. This program was described as "a software tool to file-sharing, that can be more powerful than Napster". Immediately the page was put out of service, but too many copies were unloaded. Nullsoft never published the software again.

Gnutella network is made up of a great number of gnodes distributed throughout the world. Gnutella works in a distributed environment model. A **GNODE** X (host running Gnutella software) offers the files that wishes to share with other users, simultaneously the user running the gnode Y can be recovering, as a client, files from gnode X. This is very similar to Napster. The main difference is located in its topology which does not indicate hierarchy some, since each gnode is equal to the others in functionality; there is no central-server.

Each gnode only knows about the gnodes with which it directly connects. The other gnodes are invisible, unless they announce themselves answering a message. The Gnutella Network operates under the model known like 'viral propagation". A client sends a message to a gnode, and this one send it again to all gnodes with which it is connected. Of this form, with single network address of a connected gnode, someone can enter to the network.

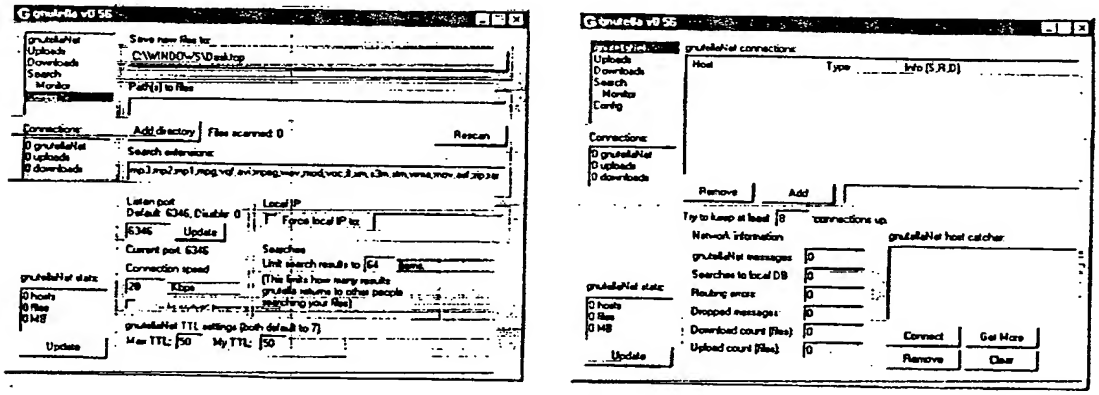

**FIGURE 2: Screenshots of the Gnutella v0.6 application**

### *Gnutella Messages Data Structures*

A Gnutella message is transported on TCP protocol. Its head always have 23 bytes with the following fields:

| Gnode ID | Function | TTL | Hops | Payload length |
|---|---|---|---|---|
| 16 Bytes | 1 Byte | 1 Byte | 1 Byte | 4 Bytes |

- ***Gnode Identifier***: the header contains a message identifier which is generated for each new message it creates. This 16 Bytes number can be considerated as unique in the Gnutella network.
- ***Function***: defines the message type of the packet (see next paragraph).
- ***TTL***: Time-To-Live. It defines over how many gnodes the packet can be forwarded before it is removed from the network. Every gnode receiving a packet decrement its TTL. If the TTL reaches zero the gnode drops the packet. When a new message is sent

from a gnode, the TTL is set to whatever the user has set in the configuration menu of his Gnutella software [Figure 2].

- ***Hops***: the number of times the packet has been forwarded. The packet is dropped, if the number of hops is equal to the "maximal hops" ( MyTTL) set in the configuration of Gnutella [Figure 2].
- ***Payload length***: the length of the Function-dependant data which follows.

The TTLs and Hops are the only mechanism for expiring packets on the network. TLL and Hops define the "search horizon" of a gnode. Abuse of the TTL field will lead to an unnecessary amount of traffic and poor network performance.

The payload length is the only reliable way for a gnode to find the beginning of the next descriptor in the input stream. Contrarily to the Napster protocol, the Gnutella protocol does not provide a synchronisation method. It has no blanks between the blocks of the message. Therefore a gnode getting out of synchronisation with its input stream should drop the connection. On the other hand Gnutella messages are extremely short.

## *Function IDs*

- ***Ping (0x00)***: empty message (payload has zero length) sent by a gnode requesting a Pong (0x01) from the other gnodes on the network.

- ***Pong (0x01):*** sent in response to a 0x00. The payload contains the host IP and port, the number of shared files and the number of kbytes shared. Such a message has this payload:

| Port<br>2 Bytes | IP Address<br>4 Bytes | Number of Files shared<br>4 Bytes | Number of Kbytes Shared<br>4 Bytes |
|---|---|---|---|

- ***Query (0x80):*** this is a search message and contains the query string as well as the minimum speed.

| Minimum Speed<br>2 Bytes | Search criteria<br>n bytes |
|---|---|

- ***Query Hit (0x81):*** the results of a 0x80 query. This packet contains the IP address, port and speed of the gnode, followed by a list of file sizes and names, and the 16 bytes client gnode identifier which found the files. The client gnode identifier was created once when the software has been installed. It never changes and is stored in the gnutella.ini.

| Number of Hits<br>1 Byte | Port<br>2 Bytes | IP Address<br>4 Bytes | Speed<br>4 Bytes | Result Set<br>n bytes | Gnode ID<br>16 Bytes |
|---|---|---|---|---|---|

**Result Set**

| File Index<br>4 Bytes | File Name<br>n bytes |
|---|---|

## Propagation Rules

Peer-to-peer model used by Gnutella requires that the gnodes have the ability to propagate received messages (Ping, Pong, Query; Query Hit) through their open connections. To operate in a efficient way, (considering network topology, the non-hierarchy and the existence of loops) gnodes must implement several propagation rules [Figure 3], such as:

- *1st Rule*: A gnode will propagate Ping and Query messages to all gnodes which it's directly connected, except the one that gave the message.
- *2nd Rule*: Pong, Query Hit messages must be propagated by the same way by which the initial associated message (Ping or Query) travelled. This is possible because before propagating a message, gnodes review in internal tables by which generated such answer, and thus obtain the connection by where it must send this again.
- *3rd Rule*: A gnode will decrement the value of field TTL in one unit and will increase the value of the Hops field in one unit before propagating this message by the pertinent connections. If when decrementing the value of TTL the obtained result is zero, it must drop the message.
- *4h Rule*: If a gnode receives a Ping or a Query message with identical gnode identifier and same function ID that one received previously (in a brief period of time, not formally specified) it should avoid to propagate it.

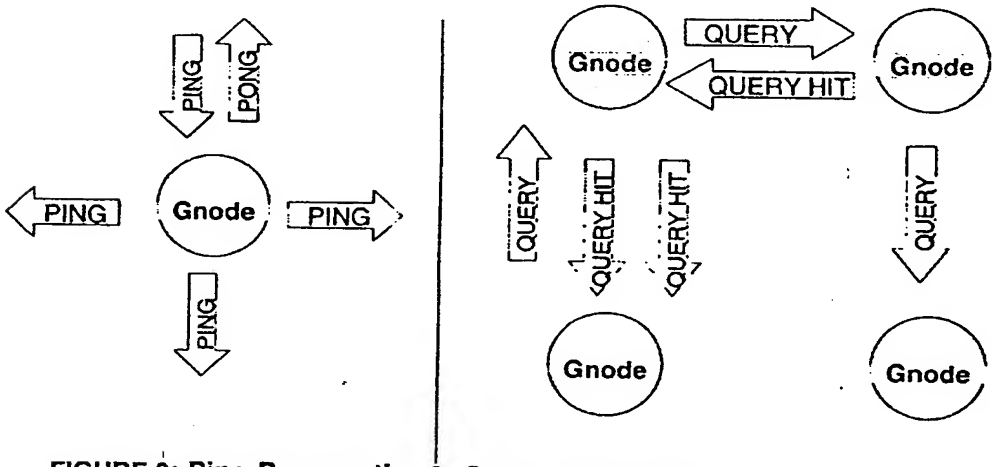


**FIGURE 3: Ping Propagatio & Query Propagation**

## Initialisation

The gnode connects itself to the network by establishing a connection with another known gnode. The acquisition of the first IP address needed to connect the network is not part of he protocol.

Once the address of another gnode on the network is obtained, a TCP/IP connection to this gnode acting like a server is created by the gnode acting like a client. The client is sending the HTTP packet **GNUTELLA CONNECT/0.56** and is expecting back a **GNUTELLA OK**. At this point the two gnodes can exchange Gnutella messages. The server is now expecting the client to identify himself with a **PING**. After this the server responds with a **PONG**, transmitting that way the number and the size of files.

**EXAMPLE:**
Four computers are running the Gnutella software and have the following characteristics:

Gnode IP address: 134.210.97.2 (only last byte in graphs)
Gnode Identifier: 4E62 (four last bytes)
Shared Files: nothing

Gnode IP address: 134.210.97.3
Gnode Identifier: 6754
Shared Files: 'p2p.doc"

Gnode IP address: 134.210.97.149
Gnode Identifier: 3760
Shared Files: "song1.mp3"

Gnode IP address: 134.210.97.150
Gnode Identifier: 4d60
Shared Files: "song2.mp3"

In the following graphs, each arrow indicates the message direction, and its identification consists of the header: Gnode ID, message type, TTL and Hops. In addition some important parts of the payload (e.g. IP addresses of hosts) are following. The heavy arrow indicates an opened connection and its direction (from the opening gnode)

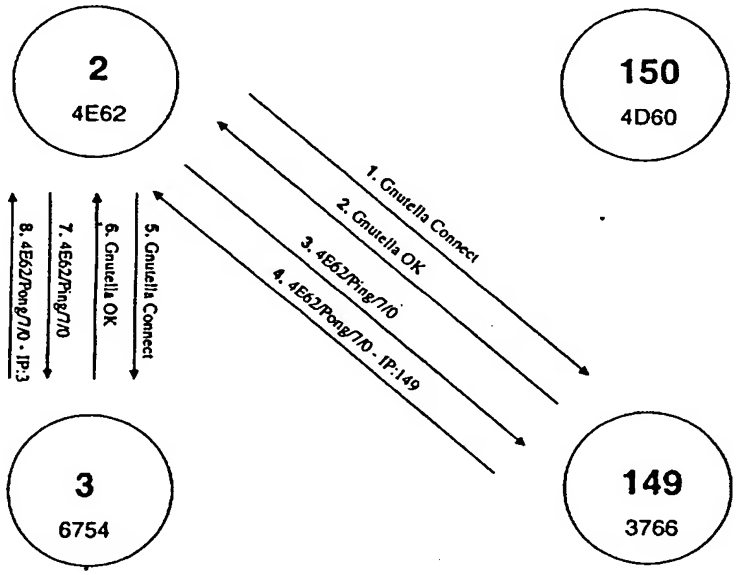


**Figure 4:**
The Gnode 2 has acquired the IP addresses of the Gnode 3 and 149. So it can open a communication on application layer with them .

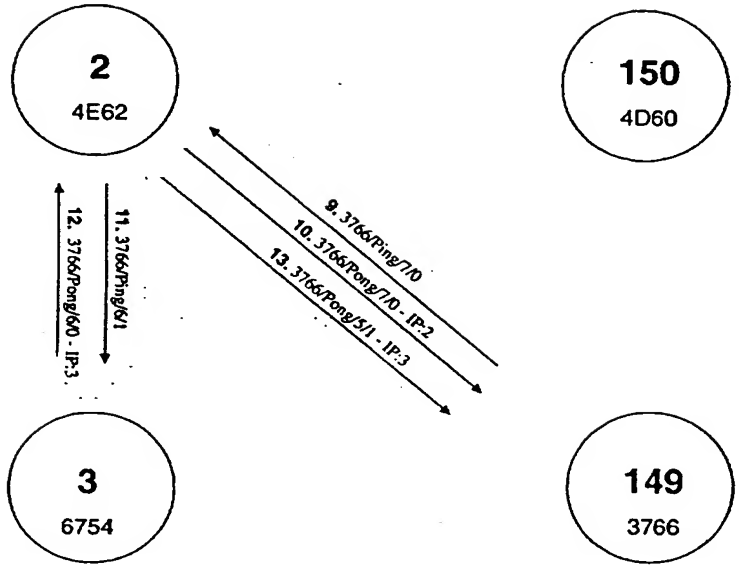


**Figure 5:**
The gnode 149 execute an "update" application command, which generates the shipment of a Ping message by all its open connections.

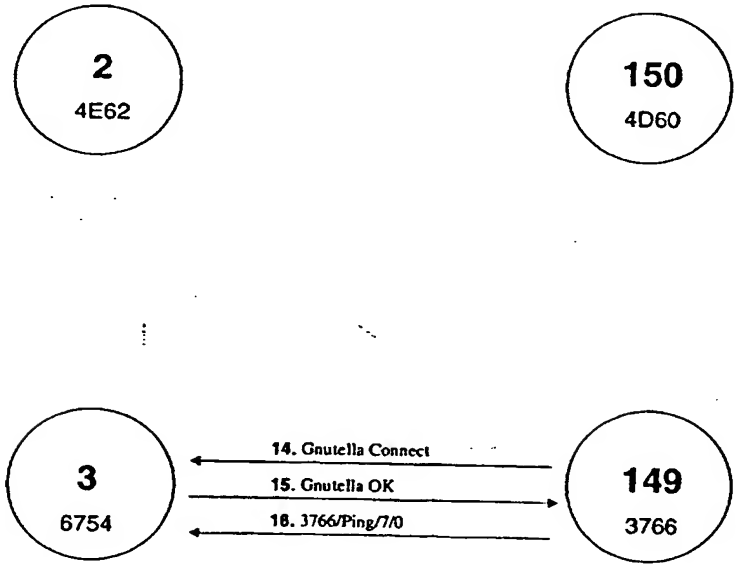


**Figure 6:**
As receiving information of the presence of Gnode 3, Gnode 149 opens a connection with it, and sends a Ping message. Gnode 3 does not propagate this Ping and does not send a Pong message back, because it has received Ping message (message 11) with same function and Gnode ID for a brief time period.

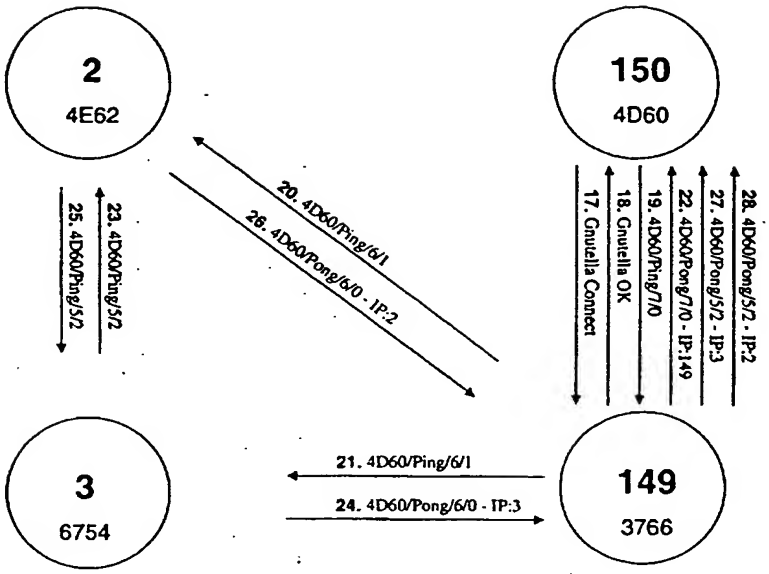


**Figure 7:**
The Gnode 150 has acquired the IP addresses of gnode 149. Thus, it opens a connection with this gnode. Gnode 150 sends a Ping message to gnode 149, which is answered with a Pong and propagated by all its open connections. The Ping messages 23 and 25 has been dropped respectively by gnodes 2 and 3, because they have received same messages from gnode 149. The answer travels by the same ways where the Ping message circulated and, finally, it arrives at gnode 150. The gnode 150 has acquired the IP addresses of all hosts of the network. The gnode 150 could open a connection with gnodes 2 and 3; but to reduce traffic and computation time it doesn't.

### *File request*

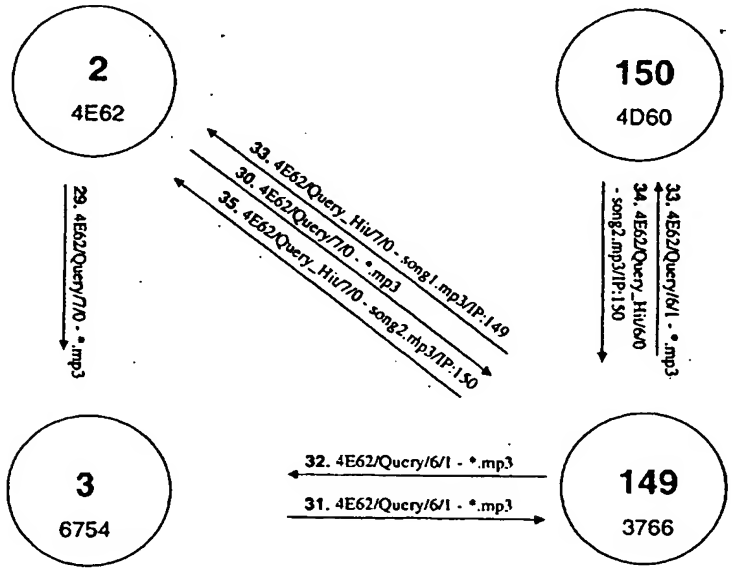


**Figure 8**

At this point the gnodes can send **QUERY** messages. Gnodes propagates these messages like **PINGs.** The gnodes, acting like servers, answer with **QUERY_HIT** messages which contains an index of all requested songs the host is sharing.

---

**Figure 8:**
Gnode 2 makes a query indicating the *.mp3 pattern. The messages 31 and 32 are not propagated because the gnodes 3 and 149 got the query message already from other gnodes. Gnode 3 doesn't answer the query because it has no files which match the search pattern. Gnodes 149 and 150 have files that match the search pattern. As a consequence they send to gnode 150 query_hit messages which include their IP and Port numbers.

### *File Transfer*

The file transfer is very similar to Napster. Files are downloaded out-of-network i.e. a direct connection between the source and target is established in order to perform the data transfer. File data is never transferred over the Gnutella network.

The file download protocol is HTTP. Gnodes initiating the download send a request string of the following form to the target server:

```
Get /get/<File Index>/<File Name>/HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\
User-Agent: Gnutella\r\n
\r\n
```

File Index/File Name are extracted from the 'Result Set" descriptor of the Query_Hit messages. The range value is needed to resume downloads at the point at which they terminated.

The gnode, acting like a server, receiving this download request responds with HTTP 1.0 compliant headers such as:

```
HTTP 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 5000000\r\n
```

The file data then follows and should be read up to the number of bytes specified in the Content-length provided in the server's HTTP response.

## 3.3 Napster & Gnutella – a comparison

Napster and Gnutella build system redundancy which can reduce remarkably the bottlenecks situation on centralised client/server networks. Files don't have to be stored in central server. Everyone running an electronic device at home, in the office or simply in his pocket (e.g. an advanced mobile phone) can share his files with other people over the internet. To achieve it, the user can choose between to different P2P concepts: the Napster or the Gnutella protocol.

The main technical difference between them is the way how they manage to get the port and IP-address of the peer sharing the file of interest. Downloading the file is very similar.

Napster is using for that issue an central index server. Advantages are:

- Peers don't have to root incoming messages. Thus peers need less computing time and bandwidth to root this messages. Peers are acting as servers just when another client wants to upload a shared file from it.
- No network flooding with queries is needed. The result is a smaller network traffic in comparison to Gnutella. The central server is answering directly the search queries. No complex routing is needed.
- The problem of a bottleneck situation in the context of the central index server is not too significant. The quantity of incoming data is not too important for a small number of high-power server. The Napster network is still scalable.

These enumerated advantages makes the Napster protocol very easy to use. On the other hand, the main problems are more of a political than of a technical dimension.

- Napster does not offer anonymity to the user. Everything the user is sharing and downloading is stored on a central server. The Napster company can creates profiles of the users.
- Napster can be shut down simply because it has a server and the site running that server can be sued for any political reason.

The centralisation of information on servers makes overall system susceptibility to single-point failure. The fear of censure and control has lead to a growing interest in decentralised P2P network such as Gnutella. Every gnode (host running Gnutella application) is running its own small index server. This lead to advantages and disadvantages:

- Considering every gnode only knows a small part of the Network, Gnutella can offer a higher degree of anonymity than Napster. On the other hand, messages including IP and port numbers are not encrypted.
- Because Gnutella doesn't run any central server, it is more difficult to shut down the network. Users of a particular network fear any establishment that could take control over 'their" network. Indeed, the music industry is trying at this time to convince ISPs (Internet Service Provider) to shut the 'Gnutella port". In fact, most of gnodes are using the default port 6346 [see Figure 2]. This operation would disturb the network until the users would have change the port number. This operation has no long time issues.
- Depending on the TTL values, search queries are flooding more or less the network. This leads to higher traffic. The scalability of the network is directly linked to the disposed bandwidth which is often very low. At the present time the majority of the gnodes are still running with a 56k modem. This have heavy issues on the network. This last point will probably loose its weight as network hosts will acquire more and more "flatrates" with broadband access.
- As the number of peers and files in Gnutella increases more and hosts are pushed outside the 'search horizon" and files held by those hosts become beyond reach.

For the future we could conceive a combination of the two presented protocols.

The P2P application would let users choose the function of their hosts: client, server or peer. Hosts with broadband access would be encourage to run a server. They would manage a "local index server". This servers would exchanged their table of content using a flexible protocol like similar to Gnutella. On the other hand, hosts with poor bandwidth would choose to act like clients. These hosts have to keep a connection with one of the servers of the network.
The reduced number of servers will increase the efficacy and scalability of the network compared to a "pure" Gnutella network. And the ease of use of a "Napster-network" is preserved. The reduced but not insignificant number of servers will guarantee a decentralised network topology.

This scenario would fit e.g. to universities who want to exchange public documents which are laying on the drives of a bunch of desktops and web-servers. Every university in the world would run one server which indexes all documents of the campus. Such a P2P network would reduce the searching time for documents and increase co-operation between universities.

# 4 IP Telephony

The implicit claim behind P2P file sharing is that redundancies are not enough. Another claim is that files of interest have to be found efficiently. In the context of location of users in the spacious internet, another P2P protocol has hinted the interest of the "Internet community". With the help of the Session Initialisation Protocol (SIP) it is possible to build up interesting features that IP-Telephony, Tele-Conferences and Instant Messaging.

## 4.1 SIP Protocol

**IP-Telephony** is at the origin of a bunch of exiting applications that will enhance the functionality of the internet. Integration of real-time multimedia communications, such as voice, with web, instant messaging and online games become conceivable. IP-Telephony differs in a number of points from the current phone system, both in terms of architecture and protocols. IP-telephony separates communication setup from reserving resources.

As the name implies, the **Session Initialisation Protocol (SIP)** is about initiation, modification and termination of interactive communication sessions between users. SIP is an application-layer control protocol that can invite participants to unicast and multicast sessions. SIP deals with communication setups of IP-telephony calls. Unlike Napster and Gnutella SIP is a protocol proposed by the IETF(Internet Engineer Task Force).

SIP supports the following features:

- ***User location***: determination of the end system to be used for communication.
- ***User capabilities***: determination of the media parameters to be used.
- ***User availability***: determination of the willingness of the called party to engage in communications.
- ***Call setup***: establishment of call parameters at both called and calling party, including transfer and termination of calls.

In the next paragraphs we will give a general outlook on the SIP routing technology.

### *Addressing*

Initiating a session requires determining where the user to be contacted is actually residing at a particular moment. A call for that user might need to ring all his "phones" (PC at work, PC at home, etc...) at once. Thus to identify the user, he has to be named. His name has to be unique in the whole network.

Since it is the most common form of user addressing in the Internet, SIP uses an email-like identifier of the form "*user@domain*", "*user@host*", "user@IPaddress" or "*phone-number@gateway*".

SIP uses these addresses as part of SIP-URLs, such as sip:marc@findeli.com. This URL may well be placed in a web page, so that clicking on the link initiates a call to that address, similar to mailto URL.

### *Network Architecture*

The SIP technology offers a large degree of freedom to allocate functions between network servers and user end systems. The SIP protocol is operating in a P2P architecture. Indeed, a called participant may either generate or receive requests. Users at the "end" of the line are running an application, the **User Agent Server**, which is operating like a peer.

Furthermore, SIP messages can traverse many servers which are:

- ***Proxy Server:*** a proxy server is a network device that is responsible for routing and delivering requests to the called party. Proxies interpret and, if necessary rewrite requests before forwarding them. A proxy can also make requests on behalf of other clients (e.g. in multicast sessions).
- ***Redirect Server:*** a redirect server is a server that accepts a SIP request, maps the address into new addresses and return these addresses to the client. The redirect server could act like a HOME AGENT of the user. Every time the user starts an IP-telephony application anywhere in the world, SIP sends a message to the redirect server, indicating the actual location.

Unlike in a Gnutella network, "end users", acting like peers, are not routing requests. In the SIP network topology it is not necessary to flood the network with requests. Indeed, the function of proxy and redirect servers is to forward SIP messages to the called party the most efficiently. (NB: SIP routing has a lot of similarities to Mobile IP, though SIP is working on application-layer.)

### *SIP Messages*

SIP messages are text based like Napster messages. This makes the protocol extensible for future changes.

SIP messages are transmitted with the UDP (unreliable) or the TCP (reliable) protocols.

Calls in SIP are uniquely identified, carried in the **Call-ID** header field in SIP messages. The call identifier is created by the creator of the call and used by all call participants. (The Call-ID has the same function than the gnode identifier in the Gnutella protocol.) The **From**

header of SIP messages indicates the connection source. The **To** field names the party who the originator wishes to contact.

SIP defines several methods, where the first manage or prepare calls and connections: **INVITE** invites a user to a call and establishes a new connection, **BYE** terminates a connection between two users in a call, and **OPTIONS** solicits information about capabilities, but does not setup a connection. **ACK** is used for reliable message exchanges for invitations. **CANCEL** terminates a search for a user. Finally, **REGISTER** conveys information about user's location to a SIP server.

### *Communication Setup*

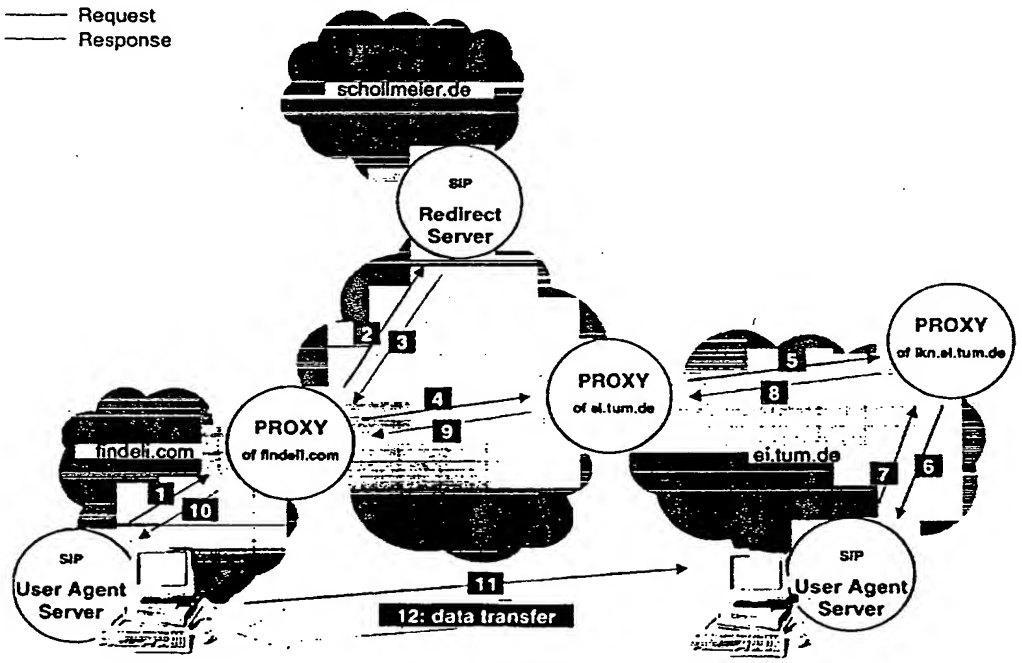


**FIGURE 9**

**EXAMPLE:**

Consider the scenario in the figure 9. The caller marc@findeli.de wishes to place a call to ruediger@schollmeier.de.

- *1:* Marc is sending an SIP **INVITE** message to the proxy of findeli.de.

  INVITE ruediger@schollmeier.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***2:*** This proxy forwards the request out to Schollmeier.de This server is actually not a proxy but a redirect server. This server looks at Ruediger in ist database, and determines that he is on ei.tum.de.

  INVITE ruediger@schollmeier.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***3:*** Therefore it sends a **REDIRECT** response to the findeli.de proxy, instructing it to try instead ruediger@ei.tum.de.

  302 Moved temporarily
  Location: ruediger@ei.tum.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***4:*** The findeli.de proxy directly tries to contact ei.tum.de. It sends an **INVITE** request to the ei.tum.de proxy.

  INVITE ruediger@ei.tum.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***5:*** This server consults its database and learns that Ruediger is in the lkn. So, it sends an **INVITE** to the lkn.ei.tum.de.

  INVITE ruediger@lkn.ei.tum.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***6:*** The lkn.ei.tum.de sends an **INVITE** message to the PC (name: 'test') where Ruediger is currently sitting. But how does the proxy knows which PC Ruediger is at? SIP defines another request, called **REGISTER**, which is used to inform a proxy of an address binding. When the user turn on his SIP client on his PC, the proxy would register the binding sip:ruediger@schollmeier.de to sip:ruediger@lkn.ei.tum.de. SIP provides many possible responses such as: ACCEPTANCE, BUSY, REJECTION, REDIRECTION, and so on.

  INVITE ruediger@test.lkn.ei.tum.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- ***7-10:*** The User Agent Server (Ruediger's PC which operates like a peer) sends a response which is forwarded back through the proxies to the original caller.

  200 OK
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

- *11:* The SIP agent sends an **ACK** (acknowledgement) to establish a direct call with Ruediger. After exchanging their capabilities, Ruediger and Marc can start talking.

  ACK ruediger@test.lkn.ei.tum.de
  FROM: marc@findeli.de
  TO: ruediger@schollmeier.de
  Call-ID: test

## 4.2 Napster & Gnutella – Alternatives to SIP?

SIP requires proxy and redirect servers. It requires to build up and configure a whole new network. It is a long and expensive process to do it. Instead of using SIP, we could use the Napster or the Gnutella protocol to initialise calls.

Using the Napster protocol would mean, that every call has to be initiated over a single central server. This server would embrace the functionalities of a whole bunch of proxies and redirect servers. This is not conceivable. Indeed, Telephony is an extremely important and sensible service. A unique server would be a single point of failure. This is not admissible.

On the other hand, the Gnutella protocol could be an alternative to SIP. Indeed, it uses an decentralised routing principle. Instead of sharing files, the Gnutella user could send queries to find the IP of a person to call. Everything would work the same way as in the file sharing scenario. Such a initialisation protocol would even work in a network with poor bandwidth. Indeed, exchanged messages are very small. No long lists of search results and queries have to be sent across the network. The biggest problem of Gnutella is the fragmentation of the network. The 'search horizon" of a gnode (host running Gnutella) is limited. Thus, Gnutella is not efficient enough to build up a reliable IP-Telephony network.

# 5. Conclusion

In future, users running a Desktop PC, a palm, or a mobile phone will get more and more power over the internet. With computing power and broadband access, which are getting every year cheaper and larger, P2P networks will become more and more efficiently. P2P networks will enhance the internet with a lot of new functionalities. People will easily create communities of common affinities where they can share files and computing time, communicate instantly via voice and video, without knowing more than a small address. 'Free" P2P networks are really appealing, although it would be a pity if this new freedoms were used mainly for piracy.

## References


P2P:

IEEE Internet Computing Vol5, No1 January/February -
2001Peering at Peer-to-Peer Computing by Munindar P.Singh

IEEE Computer Vol34, No1 January 2001 –
Face-to-Face with Peer-to-Peer Networking by David Clark

NAPSTER:

Napster Messages by drscholl@users.sourceforge.net
[http://openap.sourceforge.net/napster.txt]

How Napster Works by Jeff Tyson
[http://www.howstuffworks.com/napster?printable=1]

MD5:

The MD5 Message-Digest Algoritm by R. Rivest -
MIT Laboratory for Computer Science and RSA Data Security, Inc.

GNUTELLA:

Gnutella And the Transient Web by Kelly Truelove
[http://www.openp2p.com/lpt/705]

The Gnutella Protocol Specification v0.4 (Document Revision 1.0)
[http://dss.clip2.com/GNUTELLAPROTOCOL104.pdf]

Gnutella: Distributed System for Information Storage and Searching Model Description
by Fernando R. A. Bordignon, Gabriel H. Tolossa

Free Riding an Gnutella by Eytan Adar and Bernardo A. Huberman
[http://www.parc.xerox.com/istl/groups/iea/papers/gnutella]

SIP:

Internet Telephony: Architecture and Protocols an IETF Perspective
by Henning Schulzrinne and Jonathan Rosenberg

SIP: Session Initialisation Protocol by Handley, Schulzrinne, Schooler and Rosenberg

Computer Telephony June 2000 – SIP: A Key Component for Internet Telephony